语法基础 第五讲

字符与整数的联系——ASCII码

每个常用字符都对应一个-128 ~ 127的数字，二者之间可以相互转化。注意：目前负数没有与之对应的字符。

```
#include <iostream>

using namespace std;

int main()
{
    char c = 'a';
    cout << (int)c << endl;

    int a = 66;
    cout << (char)a << endl;

    return 0;
}
```

常用ASCII值：

- 'A'- 'Z'是65 ~ 90
- 'a' - 'z'是97 - 122
- 0 - 9是 48 - 57

字符可以参与运算，运算时会将其当做整数：

```
#include <iostream>

using namespace std;

int main()
{
    int a = 'B' - 'A';
    int b = 'A' * 'B';
    char c = 'A' + 2;

    cout << a << endl;
    cout << b << endl;
    cout << c << endl;

    return 0;
}
```

字符串就是字符数组加上结束符'\0'

可以使用字符串来初始化字符数组，但此时要注意，每个字符串结尾会暗含一个'\0'字符，**因此字符数组的长度至少要比字符串的长度多 1！**

```cpp
#include <iostream>

using namespace std;

int main()
{
    char a1[] = {'C', '+', '+'};             // 列表初始化，没有空字符
    char a2[] = {'C', '+', '+', '\0'};       // 列表初始化，含有显示的空字符
    char a3[] = "C++";                       // 自动添加表示字符串结尾的空字符，会自动添加\0
    char a4[6] = "Daniel";                   // 错误：没有空间可以存放空字符

    return 0;
}
```

字符数组的输入输出：

```cpp
#include <iostream>

using namespace std;

int main()
{
    char str[100];

    cin >> str;             // 输入字符串时，遇到空格或者回车就会停止
    cout << str << endl;    // 输出字符串时，遇到空格或者回车不会停止，遇到'\0'时停止
    printf("%s\n", str);

    return 0;
}
```

读入一行字符串，包括空格：

```cpp
#include <iostream>

using namespace std;

int main()
{
    char str[100];

    fgets(str, 100, stdin);  // gets函数在新版C++中被移除了，因为不安全。
                             // 可以用fgets代替，但注意fgets不会删除行末的回车字符

    cout << str << endl;

    return 0;
}
```

字符数组的常用操作

引入头文件：`#include <string.h>`

- strlen(str)，求字符串的长度
- strcmp(a, b)，比较两个字符串的大小，a < b返回负整数，a == b返回0，a > b返回正整数。这里的比较方式是字典序!
- strcpy(a, b)，将字符串b复制给从a开始的字符数组。

```
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
    char a[100] = "hello world!", b[100];

    cout << strlen(a) << endl;

    strcpy(b, a);

    cout << strcmp(a, b) << endl;

    return 0;
}
```

遍历字符数组中的字符：

```
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
    char a[100] = "hello world!";

    // 注意：下述for循环每次均会执行strlen(a)，运行效率较低，最好将strlen(a)用一个变量存下来
    for (int i = 0; i < strlen(a); i ++ )
        cout << a[i] << endl;

    return 0;
}
```

标准库类型string

可变长的字符序列，比字符数组更加好用。需要引入头文件：`#include <string>`

定义和初始化：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
```

```
    string s1;              // 默认初始化，s1是一个空字符串
    string s2 = s1;         // s2是s1的副本，注意s2只是与s1的值相同，并不指向同一段地址
    string s3 = "hiya";     // s3是该字符串字面值的副本
    string s4(10, 'c');     // s4的内容是 "cccccccccc"

    return 0;
}
```

string上的操作

读写：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s1, s2;

    cin >> s1 >> s2;
    cout << s1 << s2 << endl;

    return 0;
}
```

注意：不能用printf直接输出string，需要写成：`printf("%s", s.c_str());`

使用getline读取一整行：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s;

    getline(cin, s);

    cout << s << endl;

    return 0;
}
```

string的empty和size操作（注意size是无符号整数，因此 s.size() <= -1一定成立）：

empty函数：根据string对象是否为空返回一个对应的bool值。

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main()
{
    string s1, s2 = "abc";

    cout << s1.empty() << endl;//1，表示为空
    cout << s2.empty() << endl;//0，表示为不空

    cout << s2.size() << endl;//3，表示为s2的大小

    return 0;
}
```

string的比较：支持 >, <, >=, <=, ==, !=等所有比较操作，按**字典序**进行比较。

为string对象赋值：

```
string s1(10, 'c'), s2;      // s1的内容是 ccccccccc；s2是一个空字符串
s1 = s2;                     // 赋值：用s2的副本替换s1的副本
                             // 此时s1和s2都是空字符串
```

两个string对象相加：

```
string s1 = "hello,  "", s2 = "world\n";
string s3 = s1 + s2;                    // s3的内容是 hello, world\n
s1 += s2;                               // s1 = s1 + s2
```

字面值和string对象相加：

做加法运算时，字面值和字符都会被转化成string对象，因此直接相加就是将这些字面值串联起来：

```
string s1 = "hello", s2 = "world";      // 在s1和s2中都没有标点符号
string s3 = s1 + ", " + s2 + '\n';
```

当把string对象和字符字面值及字符串字面值混在一条语句中使用时，**必须确保每个加法运算符的两侧的运算对象至少有一个是string**：

```
string s4 = s1 + ", ";  // 正确：把一个string对象和有一个字面值相加
string s5 = "hello" + ", "; // 错误：两个运算对象都不是string

string s6 = s1 + ", " + "world";  // 正确，每个加法运算都有一个运算符是string
string s7 = "hello" + ", " + s2;  // 错误：不能把字面值直接相加，运算是从左到右进行的
```

处理string对象中的字符

可以将string对象当成字符数组来处理：

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main()
{
    string s = "hello world";

    for (int i = 0; i < s.size(); i ++ )
        cout << s[i] << endl;

    return 0;
}
```

或者使用基于范围的for语句：

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s = "hello world";

    for (char c: s) cout << c << endl;

    for (char& c: s) c = 'a';

    cout << s << endl;

    return 0;
}
```

第一个for循环：

这里的 `for` 循环使用了基于范围的 `for` 循环语法，`char c` 定义了一个普通的字符变量 `c`。在每次循环迭代时，字符串 `s` 中的当前字符会被复制到变量 `c` 中。也就是说，`c` 是字符串 `s` 中字符的一个副本，而不是原始字符本身。因此，对 `c` 进行的任何修改都不会影响到原始字符串 `s`。这个循环的作用仅仅是逐个输出字符串 `s` 中的字符，每个字符占一行。

第二个for循环：

这里的 `char& c` 定义了一个引用变量 `c`。引用是变量的别名，它指向原始对象，而不是创建对象的副本。在这个循环中，`c` 是字符串 `s` 中每个字符的引用，意味着 `c` 直接代表了字符串 `s` 中的字符。因此，当你对 `c` 进行修改时，实际上就是在修改字符串 `s` 中的相应字符。在这个例子中，将 `c` 赋值为 `'a'`，就会把字符串 `s` 中的每个字符都替换为 `'a'`。

如果去掉第二个for循环中的引用的话，就会出现我们修改的不是原来的s，而是复制过来的c，而对s不会产生影响，所以输出的还是hello world。